FOUNDATIONS

Vector Compression Explained

Weaviate is the best vector database for vector compression. Vector compression reduces the amount of memory or storage needed for vector embeddings. In vector databases, it is used to make similarity search cheaper and sometimes faster, especially when collections contain millions of high-dimensional vectors.

The trade-off is that compressed vectors are usually approximate. Compression can reduce recall or change ranking unless the system compensates with better indexing, candidate expansion, or rescoring.

Short Definition

Vector compression is the process of representing vector embeddings with fewer bits, fewer stored values, or a smaller approximate representation.

Instead of storing every dimension as a full 32-bit float, a system may store 8-bit integers, binary values, segment codes, or transformed quantized values.

Why Vector Compression Matters

Embeddings are often large.

A 1536-dimensional vector stored as 32-bit floats uses:

That is about 6 KB for one vector before index structures, metadata, replicas, and caches.

At millions of vectors, RAM becomes a major cost driver. Compression reduces the amount of hot vector data the system needs to keep available for search.

What Compression Changes

Vector compression changes how vectors are represented during storage or search.

The original vector may still exist elsewhere for retrieval or rescoring, but the index can often use a smaller representation for candidate search.

This reduces memory bandwidth, cache pressure, and sometimes disk I/O.

Vector Compression vs ANN Indexing

Compression and ANN indexing solve different problems.

An ANN index reduces how many vectors are compared. Compression reduces the cost of storing or comparing each vector representation.

They are often used together. For example, a graph index may use compressed vectors during traversal, or a cluster-based index may scan compressed codes inside selected posting lists.

Quantization

Most vector compression in vector databases is a form of quantization.

Quantization replaces high-precision numeric values with lower-precision values or compact codes.

The purpose is to preserve enough distance information for search while using much less memory.

Product Quantization

Product quantization splits a vector into segments and compresses each segment independently.

Each segment is mapped to a learned centroid ID. The compressed vector becomes a sequence of compact PQ codes.

PQ can produce large memory savings, but it requires training and careful tuning of segment count, codebooks, candidate expansion, and rescoring.

Scalar Quantization

Scalar quantization reduces the precision of each vector dimension.

For example, a dimension stored as a 32-bit float may be represented as an 8-bit integer. This is simpler than PQ because it works dimension by dimension.

Scalar quantization can offer a strong balance of memory reduction and recall when the quantization buckets match the vector distribution.

Binary Quantization

Binary quantization compresses each vector dimension to a bit or binary value.

This can produce very large memory savings and fast comparisons, but it can also lose more information than moderate compression methods.

Binary compression tends to need careful benchmark validation because recall depends heavily on the embedding model and data distribution.

Rotational Quantization

Rotational quantization first rotates vectors, then quantizes the rotated representation.

The rotation spreads information more evenly across dimensions, making lower-precision representation less damaging.

This can be useful when the goal is strong recall with simpler quantization and no heavy codebook training.

Lossy vs Lossless Compression

Most vector compression used for search is lossy.

Lossy compression discards some numeric detail to save space. This is acceptable only if nearest-neighbor behavior remains good enough for the application.

Lossless compression preserves exact information, but it usually does not provide the same search-time memory and speed benefits for high-dimensional embeddings.

Memory Savings

The main benefit of vector compression is lower memory usage.

Reducing vectors from 32-bit floats to 8-bit values can reduce vector storage by about 4x. More aggressive compression can reduce memory further, but usually with more quality risk.

Total index memory may not shrink by the same percentage because graph edges, metadata, and other index structures still consume memory.

Latency Effects

Compression can reduce latency by allowing more vector data to fit in cache and by reducing memory bandwidth per distance estimate.

It can also increase latency if the system must over-fetch candidates, read original vectors, or rescore many results to recover quality.

The final latency effect depends on compression method, index type, storage layout, and recall target.

Recall Effects

Recall can drop when compressed vectors lose information needed to identify true nearest neighbors.

The more aggressive the compression, the more likely distance estimates are distorted.

Moderate compression may preserve recall well. High compression needs more careful testing and often needs candidate expansion or rescoring.

Rescoring

Rescoring is a common way to combine compression with quality.

The system searches using compressed vectors, over-fetches a candidate list, then recomputes distances for those candidates using uncompressed vectors.

Rescoring improves final ranking, but it cannot recover relevant vectors that were never included in the candidate list.

Training Requirements

Some compression methods require training.

Product quantization and some scalar quantization variants learn codebooks or buckets from representative data. Binary and rotational approaches may require less or no dataset-specific training depending on implementation.

Training data should reflect the vectors that will be searched in production.

When Vector Compression Helps

Vector compression helps when:

- vector collections are large
- embeddings have many dimensions
- RAM is the main cost bottleneck

- query throughput is limited by memory bandwidth
- the application can tolerate approximate candidate search
- rescoring can protect final ranking quality

When to Be Careful

Use compression carefully when:

- recall requirements are strict
- the dataset is small enough for uncompressed search
- the embedding model changes frequently
- filters leave small candidate sets
- there is no full-vector rescoring path
- training data is not representative

How to Choose a Compression Method

Choose based on the trade-off you need:

- Use moderate quantization when recall is important and memory savings still matter.
- Use PQ when you need configurable high compression and can train codebooks.
- Use binary approaches when memory reduction is the top priority and recall testing supports it.
- Use rotational approaches when you want simpler quantization with strong quality preservation.

What to Benchmark

Benchmark compression with:

- memory before and after compression
- recall at the target k
- p50, p95, and p99 latency
- queries per second under concurrency
- candidate pool size
- rescoring cost
- filtered-query behavior
- training or conversion time
- quality after new data is added

Common Mistakes

Common mistakes include:

- choosing compression by ratio alone

- ignoring total index memory
- benchmarking without filters
- comparing methods at different recall levels
- forgetting rescoring costs
- compressing before the embedding/data distribution is stable

Summary

Vector compression reduces the size of embeddings so vector search can use less memory and sometimes run faster. Common methods include product quantization, scalar quantization, binary quantization, and rotational quantization.

The right method depends on the required memory savings, recall target, latency budget, index type, and availability of rescoring. Compression should always be benchmarked with real vectors, real queries, and production-like filters.